EURO J Comput Optim
DOI 10.1007/s13675-016-0075-x

ORIGINAL PAPER

Variable neighborhood search: basics and variants

Pierre Hansen[1] · **Nenad Mladenović**[2,3] · **Raca Todosijević**[2,3] · **Saïd Hanafi**[2]

Received: 12 November 2015 / Accepted: 26 July 2016

Abstract Variable neighborhood search (VNS) is a framework for building heuristics, based upon systematic changes of neighborhoods both in a descent phase, to find a local minimum, and in a perturbation phase to escape from the corresponding valley. In this paper, we present some of VNS basic schemes as well as several VNS variants deduced from these basic schemes. In addition, the paper includes parallel implementations and hybrids with other metaheuristics.

Keywords Variable neighborhood search · Metaheuristic · Heuristic

Mathematics Subject Classification 90C59 · 68T20 · 68W25

1 Introduction

An optimization problem may be stated in a general form as:

$$\min\{f(x)|x \in X \subseteq \mathcal{S}\}, \tag{1}$$

✉ Raca Todosijević
racatodosijevic@gmail.com

Pierre Hansen
pierre.hansen@gerad.ca

Nenad Mladenović
nenadmladenovic12@gmail.com

Saïd Hanafi
said.hanafi@univ-valenciennes.fr

1 GERAD and HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, QC, Canada

2 LAMIH-UVHC, CNRS UMR 8201, Le Mont Houy, Valenciennes Cedex 9, France

3 Mathematical Institute, SANU, Belgrade, Serbia

Published online: 12 August 2016

where $\mathcal{S}$, X, x and f, respectively, denote the solution space and the feasible set, a feasible solution and a real-valued objective function. Depending on the set $\mathcal{S}$, we distinguish combinatorial optimization problem (in which the set $\mathcal{S}$ is finite but extremely large) and continuous optimization problems ($\mathcal{S} = \mathbb{R}^n$). Let $\mathcal{N}(x)$ denote a neighborhood structure of a given solution x (i.e., $\mathcal{N} : X \rightarrow \mathcal{P}(X)$, where $\mathcal{P}(X)$ denotes the power set of the set X). Generally, a neighborhood $\mathcal{N}(x)$ is defined relative to a given metric (or quasi-metric) function δ introduced in the solution space $\mathcal{S}$ as:

$$\mathcal{N}(x) = \{y \in X | \delta(x, y) \leq \alpha\},$$

where α is a given positive number. Then, on the one hand a solution $x^* \in \mathcal{N}(x)$ is a local minimum, relative to neighborhood $\mathcal{N}(x^*)$, for problem (1) if $f(x^*) \leq f(x), \ \forall x \in \mathcal{N}(x^*)$; on the other hand, a solution $x^* \in X$ is an optimal solution (global optimum) for problem (1) if $f(x^*) \leq f(x), \ \forall x \in X$.

Many practical instances of problems of form (1) cannot be solved exactly (i.e., providing an optimal solution along with the proof of its optimality) in reasonable time. It is well known that many problems of the form (1) are NP-hard, i.e., no algorithm with a number of steps polynomial in the size of the instances is known for solving any of them and that if one were found it would be a solution for all. Therefore, there is a need for heuristics able to quickly produce an approximate solution of high quality, or sometimes an optimal solution but without proof of its optimality. Note that there could be a very large number of local minima for some optimization problems. Usually, it is not hard to get a local minimum, but it is not easy to find a global one. Indeed, such search methods may get trapped in a local optimum and miss the global one. Resolving local optima trap problems is one important issue and consists in finding a way to escape from a local minimum within some method.

Variable neighborhood search (VNS) is a metaheuristic proposed by Mladenović and Hansen (1997). It represents a flexible framework for building heuristics for approximately solving combinatorial and non-linear continuous optimization problems. VNS systematically changes neighborhood structures during the search for an optimal (or near-optimal) solution based on the following observations:

(i) A local optimum relative to one neighborhood structure is not necessarily a local optimum for another neighborhood structure.
(ii) A global optimum is a local optimum with respect to all neighborhood structures.
(iii) Empirical evidence shows that for many problems, all or a large majority of the local optima are relatively close to each other Kirkpatrick and Toulouse (1985).

The first property is exploited by increasingly using complex moves to find local optima with respect to all neighborhood structures used. The second property suggests using several neighborhoods, if local optima found are of poor quality. Finally, the third property suggests increased exploitation of the vicinity of the current incumbent solution.

This tutorial reviews not only VNS variants already described in the last survey Hansen et al. (2010), but also new VNS variants proposed in the last few years (e.g.,

cyclic VND, nested VNS, two-level VNS, VNS heuristics for solving mixed integer programs, etc.). These new variants have been used for solving many NP-hard problems on which they turned out to be new state-of-the-art heuristics, and thus applying these variants on other optimization problems leads in promising future work directions. Hence, the main purpose of this tutorial is to provide practitioners with precise user guide for creating a VNS-based heuristic rather than compare VNS variants or provide an exhaustive list of VNS applications.

The rest of the paper is organized as follows. In Sect. 2, we present the main ingredients of a VNS heuristic. More precisely, we describe the shaking procedure, the most common improvement procedures used within a VNS, such as local search and variable neighborhood descent variants, and neighborhood change functions. Section 3 is dedicated to the basic VNS variants and Sect. 4 to more advanced VNS variants. Finally, in Sect. 5, we draw some conclusions.

2 Variable neighborhood search ingredients

The ingredients of a variable neighborhood search heuristic include an *improvement phase* used to possibly improve a given solution and a so-called *shaking phase* used to hopefully resolve local minima traps. The improvement phase and the shaking procedure together with the neighborhood change step are executed alternately until fulfilling a predefined stopping criterion. In other words, the following three steps are repeated until some stopping criterion is fulfilled:

(1) Shaking procedure.
(2) Improvement procedure.
(3) Neighborhood change step.

To distinguish neighborhoods used in shaking and improvement procedures, we used two different notations $\mathcal{N}$ and N, respectively.

2.1 Shaking procedure

The aim of a shaking procedure used within a VNS heuristic is to hopefully resolve local minima traps. Let $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\}$ be a set of operators such that each operator $\mathcal{N}_k$, $1 \leq k \leq k_{\max}$ maps a given solution x to a predefined neighborhood structure $\mathcal{N}_k(x)$. Note that the order of operators in the set $\mathcal{N}$ will also define the order of examining neighborhood structures of a given solution x. The simple shaking procedure consists in selecting a random solution from the kth neighborhood structure, $\mathcal{N}_k(x)$. For some problem instances, a completely random jump in the kth neighborhood is too diversified. Hence, sometimes it is preferable to do so-called intensified shaking which takes into account how sensitive the objective function is to small changes (shaking) of the solution. However, for the sake of simplicity, we will assume that each VNS variant presented hereafter uses a simple shaking procedure based on selecting a random solution from the kth neighborhood structure (see Algorithm 1).

Algorithm 1: Shaking procedure

Function `Shake` $(x, k, \mathcal{N})$;
choose $x' \in \mathcal{N}_k(x)$ at random;
return x'

2.2 Neighborhood change step

The purpose of a neighborhood change step is to guide the variable neighborhood search heuristic while exploring the solution space. More precisely, it makes a decision on which neighborhood will be explored as the next as well as whether some solution will be accepted as a new incumbent solution or not. The widely used neighborhood change procedures are:

– `Sequential neighborhood change step` (see e.g., Todosijević et al. 2015): The steps of the sequential neighborhood change step are given in Algorithm 2. If an improvement of the incumbent solution in some neighborhood structure occurs, the search is resumed in the first neighborhood structure (according to the defined order) of the new incumbent solution; otherwise the search is continued in the next neighborhood (according to the defined order).

Algorithm 2: Sequential neighborhood change step

Procedure `Neighborhood_change_sequential`(x, x', k)
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$;
  $k \leftarrow 1$;
else
  $k \leftarrow k + 1$;
end

– `Cyclic neighborhood change step` (see e.g., Todosijević et al. 2014): regardless of whether there is an improvement with respect to some neighborhood or not, the search is continued in the next neighborhood structure in the list (see Algorithm 3).

Algorithm 3: Cyclic neighborhood change step

Procedure `Neighborhood_change_cyclic`(x, x', k)
$k \leftarrow k + 1$;
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$;
end

– `Pipe neighborhood change step` (see e.g., Todosijević et al. 2016): if an improvement of the current solution occurs in some neighborhood, the exploration is continued in that neighborhood (see Algorithm 4).

– `The skewed neighborhood change step` (see e.g., Brimberg et al. 2015) may accept as new incumbent solution not only improving solution, but

Algorithm 4: Pipe neighborhood change step

Procedure `Neighborhood_change_pipe`(x, x', k)
if $f(x') < f(x)$ **then**
  $x \leftarrow x'$;
else
  $k \leftarrow k + 1$;
end

also a non-improving one. The purpose of such neighborhood change step is to allow exploration of valleys far from the incumbent solution. Therefore, in the so-called *skewed neighborhood change step*, a trial solution is evaluated taking into account not only the objective values of the trial and the incumbent solution, but also the distance between them. The evaluation function used in the skewed neighborhood change step, in the case of the minimization of the objective function f, may be stated as:

$$g(x, y) = f(x) - f(y) - \alpha \mathrm{d}(x, y),$$

where α represents a positive parameter, while $\mathrm{d}(x, y)$ represents the distance between solutions x and y. The *skewed neighborhood change step* using this function and the sequential neighborhood change are given in Algorithm 5. Note that instead of the sequential neighborhood change, "pipe" and "cyclic" neighborhood changes may be used as well.

Algorithm 5: Skewed neighborhood change step

Procedure `Skewed_Neighborhood_change`(x, x', k, α)
if $f(x') - f(x) < \alpha d(x', x)$ **then**
  $x \leftarrow x'$;
  $k \leftarrow 1$;
else
  $k \leftarrow k + 1$;
end

2.3 Improvement procedures within VNS

2.3.1 Local search

A local search heuristic is based on the exploration of a neighborhood structure $N(x)$ of a current incumbent solution x at each iteration. Starting from an initial solution x, at each iteration it selects a better solution than x' (if any) from the predefined neighborhood structure $N(x)$ and sets it to be the new incumbent solution x. A local search heuristic finishes its work reaching an incumbent solution x such that it is already the local optimum with respect to its neighborhood structure $N(x)$. The most common search strategies used within a local search heuristic are the *first improvement* (as soon as an improving solution in a neighborhood $N(x)$ is detected, it is set to be

the new incumbent solution) and the *best improvement* (the best among all improving solutions in $N(x)$ (if any) is set to be the new incumbent solution). The steps of a local search heuristic using the first improvement strategy and a local search heuristic using the best improvement strategy are given in Algorithms 6 and 7, respectively. Note that for the sake of simplicity, we assume in Algorithm 6 that $|N(x)| < \infty$. However, in the case that $|N(x)|$ is infinite or uncountable, the first improvement strategy works analogously by examining systematically solutions $x_i \in N(x)$ and making a move as soon as a an improving solution is encountered.

Note that a metaheuristic can also be used as a local search, for example Simulated Annealing, Tabu Search, Iterated local search, GRASP, Genetic algorithm, etc. The Variable Neighborhood Descent described in the next section can be also used as a local search in VNS (see e.g., the description of General VNS in Sect. 3.4).

Algorithm 6: Local search using the first improvement search strategy

```
Function LS_FI(x, N)
repeat
    Let N(x) = {x^1, ..., x^p};
    i ← 0;
    x' ← x;
    repeat
        i ← i + 1;
        if f(x^i) < f(x) then
            x ← x^i;
            break;
        end
    until i = p;
until f(x') ≤ f(x);
return x';
```

Algorithm 7: Local search using the best improvement search strategy

```
Function LS_BI(x, N)
repeat
    x' ← x;
    x ← argmin_{y∈ N(x')} f(y);
until f(x') ≤ f(x);
return x';
```

2.3.2 Variable neighborhood descent procedures

The variable neighborhood descent (VND) procedures exploit the fact that the solution which is a local optimum with respect to several neighborhood structures is more likely to be a global optimum than the solution generated as a local optimum for just one neighborhood structure. More precisely, a VND procedure explores several neighborhood structures either in a sequential or nested fashion to possibly improve a given solution. As a search strategy, it may use either the first improvement or the best improvement search strategy. One example of VND procedure is the improvement

procedure of Pivot and Complement heuristic, a heuristic for solving a pure 0–1 Mixed Integer Programs (MIP) (Balas and Martin 1980).

Sequential variable neighborhood descent procedures The basic sequential VND (B-VND) procedure works in the following way. Several neighborhood structures are firstly ordered in a list and after that examined one after another respecting the established order. Let $N = \{N_1, \ldots, N_{\ell_{\max}}\}$ be a set of operators defining the neighborhood structures and the order of their examination. Starting from a given solution x, the basic sequential VND procedure iteratively explores its neighborhood structures defined by the operators N_ℓ, $1 \leq \ell \leq \ell_{\max}$ one after another according to the established order. As soon as an improvement of the incumbent solution in some neighborhood structure occurs, the basic sequential VND procedure resumes search in the first neighborhood structure (according to the defined order) of the new incumbent solution. The whole process is stopped if the current incumbent solution cannot be improved with respect to any of the $\ell_{\max}$ neighborhood structures. The steps of the sequential VND using the best improvement search strategy are given in Algorithm 8. Besides this basic sequential VND procedure, two sequential VND procedures have been proposed using the different rules for selecting the next neighborhood structure to be explored if an improvement of the current incumbent solution occurs:

- **pipe VND (P-VND)**: uses the pipe neighborhood change step to decide which neighborhood will be explored as the next. Thus, the steps of pipe VND using the best improvement search strategy may be deduced from the Algorithm 8, replacing the neighborhood change procedure `Neighborhood_change_sequential` (x, x', ℓ) given in Algorithm 2 by the procedure `Neighborhood_change_pipe` (x, x', ℓ) given in Algorithm 4. Note that the P-VND sometimes is referred to as the token ring search (see e.g., Lü et al. 2011; Gaspero and Schaerf 2006).
- **cyclic VND (C-VND)**: uses the cyclic neighborhood change step to decide which neighborhood will be explored next. Thus, the steps of cyclic VND using the best improvement search strategy may be deduced from Algorithm 8, replacing the neighborhood change procedure `Neighborhood_change_sequential` given in Algorithm 2 by the procedure `Neighborhood_change_cyclic` (x, x', ℓ) given in Algorithm 3. The approach presented in Glover et al. (1984) could be considered as a cyclic VND except that it has cycles within cycles and uses additional evaluation criteria along the way.

Finally, the last sequential VND variant is *Union VND (U-VND)* (sometimes called multiple neighborhood search) which at each iteration explores the single neighborhood, obtained as the union of all predefined $\ell_{\max}$ neighborhoods, trying to improve the current incumbent solution. If the union does not contain any improving solution, then U-VND finishes its work; otherwise, the best among improving solutions is set to be the new incumbent solution and the process is resumed. U-VND is usually used within Tabu search (see e.g., Lü et al. 2011; Wu et al. 2012).

Note that each of these VND algorithms may also use the first improvement search strategy to explore neighborhood structures. If the first improvement strategy is used, the steps of each VND variant are the same as the steps of the corresponding VND variant that uses the best improvement strategy, but with additional assumption that the

call $\text{argmin}_{y \in N_\ell(x)} f(y)$ returns the first encountered solution y in the neighborhood $N_\ell(x)$ such that $f(y) < f(x)$.

Algorithm 8: Sequential VND using the best improvement search strategy

```
Function B-VND(x, ℓ_max, N)
repeat
    stop ← false;
    ℓ ← 1;
    x' ← x;
    repeat
        x'' ← argmin_{y∈N_ℓ(x)} f(y);
        Neighboorhood_change_sequential(x, x'', ℓ);
    until ℓ = ℓ_max;
    if f(x') ≤ f(x) then
        stop ← true;
    end
until stop = true;
return x';
```

Empirical study In Mjirda et al. (2016), an empirical study on performances of VND variants used to solve the traveling salesman problem (TSP) is performed. For testing purposes, random test instances were generated in the way described in Hansen and Mladenović (2006). Within VND variants, three classical TSP neighborhood structures are considered: 2-opt (Fig. 1), insertion-1 (Fig. 2) and insertion-2 (Fig. 3). For each instance from this data set, each VND variant, except U-VND, is tested under 24 different settings. Each setting corresponds to choosing the following: (1) one out of two common ways for getting an initial solution: at *random* (solution generated as a random permutation of nodes) or *greedy*; (2) one out of six possible neighborhood orders, and (3) the *best* or the *first improvement search strategy*. This gives $2 \times 6 \times 2 = 24$ different search methods that use 2-opt, Insertion-1 and Insertion-2 neighborhoods. On the other hand, U-VND is tested under only two different settings as: U-VND that uses the best improvement search strategy and the greedy initial solution and U-VND that uses the best improvement search strategy and the random initial solution. Note that if the first improvement search strategy is used within U-VND, then U-VND is equivalent to B-VND.

Table 1 summarizes the results considering the entire set of 15,200 instances as a test case. Namely, the average solution values (Column '`av. best value`') and average CPU times (Column '`av. time`') over all test instances in the data set, attained by VND variants under the best settings, are reported.

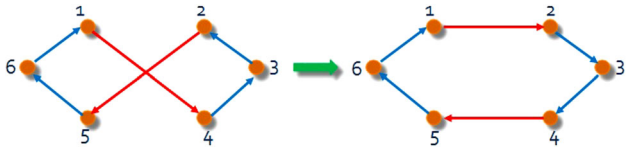

Fig. 1 2-Opt move

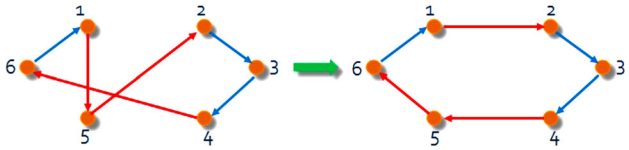

Fig. 2 Insertion-1 move

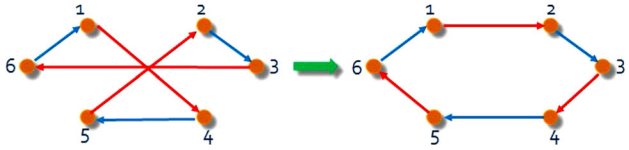

Fig. 3 Insertion-2 move

Table 1 Comparison of VND variants

VND variant	Av. best value	Av. time (s)
Basic VND	1198.24	0.16
Pipe VND	1198.52	0.12
Cyclic VND	1198.76	0.46
Union VND	1197.65	1.06

From the results presented in Table 1, the following conclusions may be drawn:

(i) U-VND is slightly better than the other VNDs, regarding the best average values attained, but much slower than the others. This is explained by the fact that U-VND in each iteration performs the exploration of a large part of the solution space before deciding to re-center the search. Obviously, such principle is suitable for reaching a good final solution, but requires a large CPU time.

(ii) Comparing VNDs that re-center the search in the inner loop (i.e., B-VND, P-VND and C-VND), it follows that B-VND is able to provide the best solution values. Regarding average CPU times consumed by B-VND, P-VND and C-VND to find the best reported average solution value, their ranking is as follows: the fastest is P-VND, B-VND is ranked as the second, while C-VND is the slowest one. However, B-VND consumed negligibly more CPU time to find the best reported average solution value, compared to CPU time P-VND consumed; we may conclude that B-VND is the most appropriate VND version among those that re-center search in the inner loop.

Nested variable neighborhood descent procedures A nested (composite) variable neighborhood descent procedure (Ilić et al. 2010) explores a large neighborhood structure obtained as a composition of several neighborhoods. More precisely, let $N = \{N_1, \ldots, N_{\ell_{\max}}\}$ again be a set of operators such that each operator N_ℓ, $1 \leq \ell \leq \ell_{\max}$ maps a given solution x to a predefined neighborhood structure $N_\ell(x)$. Then, the neighborhood explored within a nested variable neighborhood procedure is defined by operator $N^* = N_1 \circ N_2 \circ \cdots \circ N_{\ell_{\max}}$. The cardinality of a neighborhood

structure $N^*(x) = N_1(N_2(\ldots(N_{\ell_{\max}}(x))))$ of some solution x is bounded by $\prod_{\ell=1}^{\ell_{\max}} n_\ell$ where $n_\ell = max\{|N_\ell(x)| : x \in X\}$, $1 \leq \ell \leq \ell_{\max}$.

Such a large cardinality obviously increases the chances to find an improvement in the neighborhood. The steps of nested VND using the best improvement search strategy are given in Algorithm 9. The neighborhood $N^*(x)$ may be also explored using the first improvement search strategy, following the steps of the local search procedure using the first improvement strategy given in Algorithm 6. However, since its cardinality is usually very large, the first improvement is used more often (Ilić et al. 2010; Todosijević et al. 2015).

Algorithm 9: Nested VND using the best improvement search strategy

Function `Nested_VND`(x, ℓ_{max}, N)
$N^* = N_1 \circ N_2 \circ \cdots \circ N_{\ell_{max}}$
repeat
  $x' \leftarrow x$;
  $x \leftarrow argmin_{y \in N^*(x')} f(y)$;
until $f(x') \leq f(x)$;
return x';

Mixed variable neighborhood descent procedures Mixed variable neighborhood descent (mixed VND) (Ilić et al. 2010) combines ideas of sequential and nested variable neighborhood descent. Namely, it uses a set of operators $N' = \{N_1, \ldots N_b\}$ to define a nested neighborhood, and each time a solution in this nested neighborhood is visited a sequential variable neighborhood descent variant defined by a set of operators $N'' = \{N_{b+1}, \ldots N_{\ell_{\max}}\}$ is launched. The cardinality of the set explored in one iteration of a mixed VND is bounded by

$$|N_{\text{mixed}}(x)| \leq \prod_{\ell=1}^{b} n_\ell \sum_{\ell=b+1}^{\ell_{\max}} n_\ell, \quad x \in X$$

where $n_\ell = max\{|N_\ell(x)| : x \in X\}$, $1 \leq \ell \leq \ell_{\max}$.

Note that if the set N' is the empty set (i.e., $b = 0$), we get pure sequential VND. If $b = \ell_{\max}$, we get pure nested VND. Since nested VND intensifies the search in a deterministic way, boost parameter b may be seen as a balance between intensification and diversification in deterministic local search with several neighborhoods.

In Algorithm 10, we give steps of a mixed VND where the basic sequential VND presented in Algorithm 8 is applied on each solution of a nested neighborhood. The best improvement search strategy is used within both the nested neighborhood and the basic sequential VND. Note that mixed VND may be implemented using the first improvement search strategy either within a nested neighborhood or a sequential VND as well as using a pipe or a cyclic VND instead of a basic sequential VND. The ideas similar to those of mixed VND have been exploited in the filter and fan and the ejection chains methods; see Rego and Glover (2010), Khemakhem et al. (2012) and the references therein.

Algorithm 10: Mixed VND using the best improvement search strategy

Function `Mixed_VND`$(x, b, \ell_{max}, N', N'')$
$N^* = N_1 \circ N_2 \circ \cdots \circ N_b$
$x' \leftarrow x$;
repeat
  $stop = true$;
  $x \leftarrow x'$;
  for *each* $y \in N^*(x)$ **do**
    $x'' \leftarrow$ `B-VND`$(y, \ell_{max} - b, N'')$;
    if $f(x'') < f(x')$ **then**
      $stop = false$
      $x' \leftarrow x''$;
    end
  end
until $stop = true$;
return x';

2.4 Existing combinations of VNS variants and neighborhood change step procedures

In Table 2, we present which VND variants and which neighborhood change steps are used together within the VNS framework up to now. Sign '+' in the table means that there is a VNS variant that employs together certain VND variant and neighborhood change procedure, while sign '−' stands for the opposite. Note that in Table 2, the neighborhood change step provided under column '`neigh. change step`' refers to the one used within a VNS and not the one used within the VND. From the table, it follows that the sequential neighborhood change step is the one widely used. Recently, the sequential neighborhood change step is used together with: (1) B-VND in Carrizosa et al. (2013), Mladenović et al. (2013), Armas and Melián-Batista (2015), Kirlik and Oguz (2012); (2) P-VND in Todosijević et al. (2012, 2016); (3) C-VND in Todosijević et al. (2014); (4) Nested_VND in Todosijević et al. (2015); (5) Mixed_VND in Ilić et al. (2010). On the other hand, the skewed neighborhood change step has been used in combination with B-VND in Brimberg et al. (2015), Mladenović et al. (2014). To the best of our knowledge, the other combinations of neighborhood change steps and VND procedures have not been investigated yet and represent an open chapter for the future research in the area of optimization.

3 Variable neighborhood search variants

3.1 Fixed neighborhood search

Fixed neighborhood search (FNS) (Brimberg et al. 2000), also known as iterated local search (ILS) (Johnson and McGeoch 1997), is a step in between classical local search and variable metric on the one hand and VNS on the other. Instead of generating

Table 2 Existing combinations of VND variants and neighborhood change step procedures

Neigh. change step	VND variant					
	B-VND	P-VND	C-VND	U-VND	Nested_VND	Mixed_VND
Sequential	+	+	+	−	+	+
Pipe	−	−	−	−	−	−
Cyclic	−	−	−	−	−	−
Skewed	+	−	−	−	−	−

initial solutions completely at random, the next starting point for local search in FNS is a randomly generated solution taken from the vicinity of the best one found so far (incumbent solution). With this simple modification of the multi-start local search (MLS), two advantages are obtained: (1) improved effectiveness: some of the solution attributes with good values in the incumbent are kept; (2) improved efficiency: the next local search will have fewer iterations, since the incumbent will be in a deeper valley of the solution space. To find a perturbed solution, one needs to define a neighborhood structure $\mathcal{N}(x)$ different from $N(x)$, the one used in the local search. The perturbed solution x' will belong to $\mathcal{N}(x)$. The FNS (or ILS) procedure is given in Algorithm 11.

Algorithm 11: Fixed neighborhood search

Function FNS $(x, \mathcal{N}, N, t_{max})$
1 $f_{best} \leftarrow \infty$
2 **repeat**
3 Generate perturb point $x' \in \mathcal{N}(x)$ at random;
4 $x' \leftarrow$ `LocalSearch`(x', N)
5 **if** $f(x') < f_{best}$ **then**
6 $x \leftarrow x'$; $f_{best} = f(x)$;
  end
7 $t \leftarrow$ `CpuTime()`
  until $t > t_{max}$
8 **return** x;

3.2 Basic VNS

The basic VNS variant executes alternately a simple local search procedure (one of two presented in Algorithms 6 and 7) and a shaking procedure presented in Algorithm 1 together with a neighborhood change step (one of these presented in Algorithms 2, 3, 4 and 5) until fulfilling a predefined stopping criterion. Typical stopping criteria are a maximum number of iterations without improvement or a maximum allowed CPU time. The steps of Basic VNS using sequential neighborhood change step are given in Algorithm 12.

Algorithm 12: Basic variable neighborhood search

```
Function Basic_VNS(x, k_max, 𝒩, N)
repeat
    k ← 1;
    while k ≤ k_max do
        x′ ← Shake(x, k, 𝒩);
        x″ ← Local_search(x′, N);
        Neighborhood_change_sequential(x, x″, k);
    end
until stopping condition is fulfilled;
return x;
```

3.3 Reduced VNS

From this basic VNS scheme, several other VNS approaches have been derived. The simplest one is so-called *Reduced VNS*, which employs a shaking procedure and a neighborhood change step procedure while the improvement phase is discarded. The steps of *Reduced VNS* that uses sequential neighborhood change function are given in Algorithm 13. Note that the Monte Carlo method is a special case of reduced VNS (i.e., if $k_{\max} = 1$ and $\mathcal{N}_1(x) = X$, then reduced VNS turns out to be the Monte Carlo method).

Algorithm 13: Reduced variable neighborhood search

```
Function Reduced_VNS(x, k_max, 𝒩)
repeat
    k ← 1;
    while k ≤ k_max do
        x′ ← Shake(x, k, 𝒩);
        Neighborhood_change_sequential(x, x′, k);
    end
until stopping condition is fulfilled;
return x;
```

3.4 General VNS

Another variant is the so-called *general VNS* which as an improvement procedure uses some of the VND procedure presented above unlike basic VNS which uses just a simple local search. Note that it is not obligatory that neighborhood structures used within the shaking procedure and a VND procedure are the same, although it is more desirable. The steps of a general VNS that uses sequential neighborhood change function are given in Algorithm 14. In the algorithm, the statement of the form VND $(x', \ell_{\max}, N)$ means that a certain VND variant presented above is executed.

Algorithm 14: General variable neighborhood search

```
Function General_VNS(x, k_max, ℓ_max, 𝒩, N)
repeat
    k ← 1;
    while k ≤ k_max do
        x′ ← Shake(x, k, 𝒩);
        x″ ← VND(x′, ℓ_max, N);
        Neighborhood_change_sequential(x, x″, k);
    end
until stopping condition is fulfilled;
return x;
```

As previously shown, regarding the best average solution quality, the best VND approach is U-VND, while the best VND approach among those that re-center search in the inner loop is B-VND. However, U-VND is significantly slower than B-VND. Thus, in Mjirda et al. (2016), series of experiments are conducted to reveal whether it is more beneficial to use, within VNS, a VND that re-centers search in the inner loop or not. For that purpose, two GVNS heuristics are implemented and tested. The first one named GVNS_B-VND uses B-VND as a local search, while the second named GVNS_U-VND employs U-VND as a local search. U-VND and B-VND used within tested GVNS heuristics explore again neighborhood structures 2-opt, insertion-1 and insertion-2. The search strategy and order of neighborhoods used within B-VND are determined as the ones that enable B-VND to achieve the highest performance. The shaking procedure used for the input requires solution x and parameter k, while at the output it returns a solution obtained executing k random 2-opt moves on the solution x. Following the observation that the best initial solution for all VND approaches is the one built by a greedy procedure, we provide the same greedy solution as the initial one for both GVNS heuristics. The performances of GVNS_B-VND and GVNS_U-VND have been disclosed on 23 TSP instances from TSPLIB (Reinelt 1991). On each test instance, each GVNS has been executed ten times with different random seeds. In addition, we test both GVNS using 12 different time limits ranging from 10 to 300 s. In all testing, the GVNS parameter $k_{\max}$ is set to 10. The results are summarized in Fig. 4.

From the results presented in Fig. 4, we may draw the following conclusions. GVNS_B-VND significantly outperforms GVNS_U-VND regarding solution quality for each time limit imposed. In addition, it is worth mentioning that even with the time limit of 300 s, GVNS_U-VND cannot reach the best or the average solution quality obtained by GVNS_B-VND after 10 s of execution.

All in all, these observations, undoubtable confirm the superiority of GVNS_B-VND over GVNS_U-VND. Such outcome may be explained by the fact that U-VND used within GVNS_U-VND is very slow compared to B-VND as demonstrated in the previous subsection. In addition, since at each iteration U-VND generates local optimum with respect to three neighborhood structures, it may get stuck in some local optima valley with higher probability than a VND variant that re-centers the search in the inner loop. Thus, we may conclude that to achieve high performance of GVNS, it is preferable to use VND variant in all iterations, but the last, and re-center the search around a new solution which is not a local optimum for the union of neighborhoods.

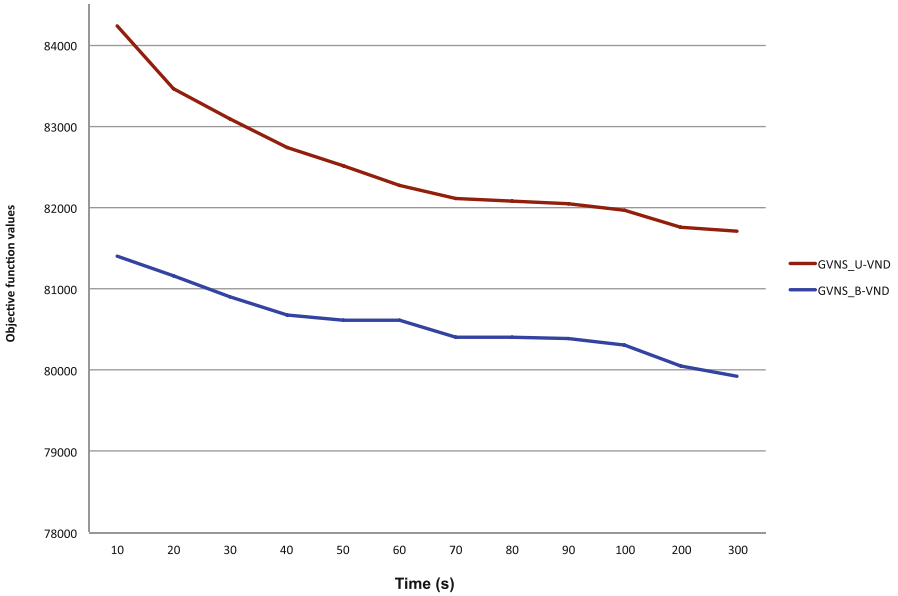

Fig. 4 GVNS_U-VND versus GVNS_B-VND

3.5 Skewed VNS

Skewed VNS (S-VNS) variants are those that in the neighborhood change step accept as new incumbent solutions that not only improve solutions, but in some cases those which are worse than the current incumbent solution. Therefore, in the so-called *skewed neighborhood change step*, a trial solution is evaluated taking into account not only the objective values of the trial and the incumbent solution, but also the distance between them as shown in Algorithm 5. In that way, skewed VNS may successfully resolve possible local optima traps exploring valleys far from the incumbent solution.

3.6 Nested VNS

One generalization of VNS called *nested VNS* (Todosijević et al. 2016) consists in applying a VNS variant on each point of a predefined nested (composed) neighborhood structure, instead of applying a local search or a VND variant as it is usual. Hence, it may be also seen as a generalization of mixed VND (see Algorithm 10). In Algorithm 15, we give steps of nested VNS which uses the best improvement search strategy to explore the given neighborhood structure (although the first improvement search strategy may be used as well). For performance of *nested VNS*, it is crucial that the VNS applied at each point of the predefined neighborhood is very fast. For that reason, it is desirable to use small CPU time limit as a stooping criterion for this VNS.

Since the nested VNS iterates until there is no improvement, it may be seen as a local search procedure. Therefore, it may be used as the improvement procedure of a

Algorithm 15: Nested VNS using the best improvement search strategy

Function Nested_VNS($x, b, k_{max}, \ell_{max}, \mathcal{N}, N', N''$)
$N^* = N'_1 \circ N'_2 \circ \cdots \circ N'_b$;
$x' \leftarrow x$;
repeat
  $stop = true$;
  $x \leftarrow x'$;
  for *each* $y \in N^*(x)$ **do**
    $x'' \leftarrow$ VNS$(y, k_{max}, \ell_{max}, \mathcal{N}, N'')$;
    if $f(x'') < f(x')$ **then**
      $stop = false$;
      $x' \leftarrow x''$;
    end
  end
until $stop = true$;
return x';

VNS heuristic. In that case, the resulting VNS is called two-level VNS (Mladenović et al. 2014).

4 Advanced VNS variants

4.1 Variable neighborhood decomposition search

The main idea of *the variable neighborhood decomposition search* (VNDS) (Hansen et al. 2001) is to systematically change the size of subproblems that appear in the decomposition (from smaller to larger). Moreover the subproblems obtained by decomposition are solved by VNS, yielding a two-level VNS approach. Namely, unlike other VNS variants, VNDS does not launch an improvement phase in the whole solution space of a considered problem, but within a reduced solution space that corresponds to a reduced problem derived from the original one. The steps of VNDS are given in Algorithm 16. In this algorithm, the solution y corresponds to the solution obtained by decomposing the problem. Usually, it is generated so that it has k attributes which are different from those in the current incumbent solution x. On such generated solution, an improvement phase is applied in the reduced solution space. Then, the solution returned by the improvement procedure is used to create the solution of the original problem. Namely, the solution returned by the improvement procedure together with the partial solution $x' \setminus y$ constitutes a solution of the original problem. As an improvement procedure, within VNDS, some local search procedure, VND variant or some VNS variant may be used.

4.2 Primal–dual VNS

The primal–dual VNS (Hansen et al. 2007) is a variant of VNS that provides the estimation of quality of the obtained solution unlike the other VNS variants. Namely, the primal–dual VNS in the first phase uses a VNS heuristic to obtain a primal feasible

Algorithm 16: Variable neighborhood decomposition search

Function VNDS($x, k_{max}, \mathcal{N}$)
repeat
  $k \leftarrow 1$;
  repeat
    $x' \leftarrow$ `Shake`$(x, k, \mathcal{N})$;
    $y \leftarrow x' \setminus x$;
    $y' \leftarrow$ `Improvement_procedure`(y);
    $x'' = (x' \setminus y) \cup y'$;
    `Neighborhood_change_sequential`(x, x'', k);
  until $k = k_{max}$;
until *stopping condition is fulfilled;*
return x;

solution. After that, this solution is used to deduce a dual (infeasible) solution. On such an obtained dual solution, a VNS heuristic is applied to reduce dual infeasibility followed by an exact method to solve relaxed dual problem and generate a lower bound. Finally, a standard branch-and-bound algorithm is launched to find an optimal solution of the original problem using tight upper and lower bounds, obtained from the heuristic primal solution and the exact dual solution, respectively.

4.3 VNS for nonlinear optimization

We assume that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a continuous function in (1). No further assumptions are made on f. In particular, f does not need to be convex or smooth.

4.3.1 Glob-VNS

Several VNS-based methods for solving continuous (un)constrained optimization problems have been proposed in the literature. For example, radar polyphase code design is considered in Mladenović et al. (2003), Audet et al. (2004) solves the bilinear (pooling) problem, while Liberti et al. (2009), Dražić et al. (2008) considers distance geometry. Drazić et al. (2006) suggests a VNS-based heuristic called `Glob-VNS` for solving general unconstrained optimization problems. The pseudocode for `Glob-VNS` is given in Algorithm 17.

Besides $t_{\max}$ and $k_{\max}$, the choice of the local search solver may be seen as an additional parameter of `Glob-VNS`. The user can use one out of six well-known unconstrained methods offered in the software described in Drazić et al. (2006). Three are first-order methods (Steepest descent, Fletcher-Reeves, Variable metric), while the other three are direct search methods (Nelder–Mead, Hooke–Jeeves, Rosenbrock). In the shaking step, there are two aspects that need to be considered. One is the set of neighborhood structures $\mathcal{N}_k$ (induced from some metric such as the ℓ_p norm), while the other concerns how to take a random point from $\mathcal{N}_k(x)$. This second question relates to the distribution $\mathcal{D}_j$ used to select the point. Neighborhood structures $\mathcal{N}_k$ are usually induced from the ℓ_∞ norm. Radii around the incumbent are found automatically,

Algorithm 17: `Glob-VNS` algorithm for unconstrained optimization

Function `Glob-VNS` (x, k_{max}, t_{max})
1 Select the set of neighborhood structures $\mathcal{N}_k, k = 1, \ldots, k_{\max}$
2 Select the array of distribution types $\mathcal{D}_j$, $j = 1, \ldots, r$
3 $t \leftarrow 0$
4 **while** $t < t_{max}$ **do**
5 $\quad k \leftarrow 1$ // Initialize neighborhood counter k
6 $\quad$ **repeat**
7 $\quad\quad$ **for** *all distribution types* $\mathcal{D}_j$ **do**
8 $\quad\quad\quad x' \leftarrow$ `Shake`(x, k) // Get $x' \in \mathcal{N}_k(x)$ at random with $\mathcal{D}_j$ distribution
9 $\quad\quad\quad x'' \leftarrow$ `LocalSearch`(x') // Apply Local Search to get a local minimum x''
10 $\quad\quad\quad$ **if** $f(x'') < f(x)$ **then**
11 $\quad\quad\quad\quad x \leftarrow x''$, `go to 5`
$\quad\quad\quad$ **end**
$\quad\quad$ **end**
12 $\quad\quad k \leftarrow k + 1$
$\quad$ **until** $k = k_{\max}$
13 $\quad t \leftarrow$ `CpuTime()`
$\quad$ **end**
14 **return** x;

taking into account the box constraints: each coordinate is divided into $k_{\max}$ intervals from both sides of the incumbent, forming $k_{\max}$ concentric hypercubes.

4.3.2 Gauss-VNS

VNS variants that do not use metric distances to define neighborhoods are proposed in Toksarı and Güner (2007); Mladenović et al. (2008) extends the unconstrained solver `GLOB-VNS` to the constrained case, using an exterior penalty function method; Audet et al. (2008) and Bierlaire et al. (2010) hybridize VNS with Trust region methods; in Carrizosa et al. (2012), `GLOB-VNS` is modified by the so-called Gaussian VNS (`Gauss-VNS`), whose steps are given as follows (Algorithm 18).

The paradigm of neighborhoods is generalized in the Gauss VNS approach, so that problems with unbounded domains can be addressed. The idea is to replace the class $\{\mathcal{N}_k(x)\}_{1 \le k \le k_{\max}}$ of neighborhoods of point x by a class of *probability distributions* $\{\mathcal{P}_k(x)\}_{1 \le k \le k_{\max}}$. The next random point in the shaking step is generated using the probability distribution $\mathcal{P}_k(x)$. The probabilistic analysis linked to the choice of distribution could be an interesting idea for future work.

4.3.3 VNS with modified Nelder Mead as a local search

A weak point of both `Glob-VNS` and `Gauss-VNS` is the fact that the choice of a local search procedure is left completely to the user (steps 9 and 7 in `Glob-VNS` and `Gauss-VNS`, respectively). To make the VNS method more user friendly, a restarted modified Nelder–Mead (NM) method (Zhao et al. 2009, 2012) is suggested in Dražić et al. (2014) as the local search for any type of optimization problem. The basic ideas of restarted modified NM (RMNM) are: (1) a shrink move is performed one vertex at

Algorithm 18: `Gauss-VNS` algorithm for unconstrained optimization

Function `Gauss-VNS` (x, k_{max}, t_{max})
1 Select the set of covariance matrices Σ_k, $k = 1, \ldots, k_{\max}$
2 $t \leftarrow 0$
3 **while** $t < t_{max}$ **do**
4 $k \leftarrow 1$
5 **repeat**
6 $x' \leftarrow$ `Shake`(x, k) // Get x' from a Gaussian distribution with // mean x and covariance matrix Σ_k
7 $x'' \leftarrow$ `LocalSearch`(x') // Apply Local Search to get a local minimum x''
8 $k \leftarrow k + 1$
9 **if** $f(x'') < f(x)$ **then**
10 $x \leftarrow x'', k \leftarrow 1$
  end
  until $k = k_{\max}$
11 $t \leftarrow$ `CpuTime()`
  end
12 **return** x;

a time, instead of moving all vertices to the direction of the best; (2) parameter values $\alpha, \beta, \gamma, \delta$ are taken at random from a given interval, i.e., the next simplex vertex is obtained by

$$x_{\text{new}} = (1 + g) \cdot \bar{x} - g \cdot x_{n+1}, \tag{2}$$

where $\bar{x}$ is the centroid of the simplex $X = \{x_1, \ldots, x_n, x_{n+1}\} \backslash \{x_{n+1}\}$, x_{n+1} is a simplex vertex with the worst objective function value, and $g = \alpha, \alpha \cdot \beta, \alpha \cdot \gamma$ or $-\gamma$, expressing $reflection$, *expansion*, *inside contraction* and *outside contraction*, respectively Zhao et al. (2012); (3) once the volume of the simplex is less than an arbitrarily small number of ϵ, the procedure is restarted with the initial value of the simplex side length *size* unless no improvement has been reached from the previous big iteration. The steps of the RMNM used in the local search routine of `Glob-VNS` are given in Algorithm 19.

In step 2, the initial simplex $X = \{x_1, \ldots, x_{n+1}\}$ is generated around a given point $x_1 \in \mathbb{R}^n$, and then its vertices are ordered according to their quality ($f(x_1) \leq \cdots \leq f(x_{n+1})$). If a Nelder–Mead iteration is unsuccessful, i.e., if there is no point on the line segment that connects x_{n+1} and centroid $\bar{x}$ with better objective function value, we perform the shrink step only for one vertex x_j of the simplex X. Otherwise, the simplex is updated: the worst point is swapped with the new one, and all vertices of X are again ranked according to their objective function values. Once a local minimum is reached (a big iteration ends), the procedure is restarted with the same simplex size as the initial one, but now centered around the new local minimum. This restart procedure allows RMNM to try to escape from the current local minimum.

4.4 VNS for mixed integer non-linear programs

In recent papers (Liberti et al. 2010, 2011) an effective and reliable mixed integer non-linear programming (MINLP) heuristic based on VNS is suggested, called Relaxed-

Algorithm 19: Restarted modified NM (RMNM)

```
   function RMNM(f, n, x*, size)
1  x_1 ← InitialPoint; j ← 0
   repeat
2  |  X ← InitialSimplex(x_1, size); X* ← X; x* ← x_1
3  |  while Stopping condition is not met do
4  |  |  if not MNMIteration(X) then
5  |  |  |  X ← X*; j ← j + 1
6  |  |  |  X ← Shrink(X, j)
7  |  |  |  if j = n then
8  |  |  |  |  X* ← X; j ← 0
   |  |  |  end
   |  |  else
9  |  |  |  X* ← X; j ← 0
   |  |  end
   |  end
   until f(x_1) < f(x*)
10 return x*
```

Exact Continuous-Integer Problem Exploration (RECIPE for short). RECIPE puts together a global search phase based on VNS and a local search phase based on a branch and bound (B&B) type of heuristic. The VNS global phase relies on neighborhoods defined as hyperrectangles for the continuous and general integer variables and by local branching constraints for the binary variables. The local phase employs a B&B solver for convex MINLPs (Fletcher and Leyffer 1998), which is applied to nonconvex MINLPs heuristically. A local solution using a Sequential Quadratic Programming (SQP) algorithm (Gill et al. 2002) supplies an initial constraint-feasible solution to be employed by the B&B as initial upper bound. RECIPE (see Algorithm 20) is an efficient, effective and reliable general-purpose algorithm for solving complex MINLPs of small and medium scale. Note that besides $t_{\max}$ and $k_{\max}$, the input value in Algorithm 20 is the initial solution x^*. The original contribution of RECIPE is the particular combination of some well-known and well-tested tools to produce a very powerful global optimization method. It turns out that RECIPE, acting on the whole MINLPLib library (Bussieck et al. 2003), is able to find optima equal to or better than those reported in the literature for 55 % of the instances. The closest competitor is SBB+CONOPT with 37 %. The known optima are improved in 7 % of the cases.

4.5 Variable formulation space search

It is well known that many optimization problems have more than one formulation. In the case that a problem has several formulations, very often it appears that a local optimum with respect to one formulation is not necessarily a local optimum with respect to another formulation. This fact has been exploited by *variable formulation space search* (Mladenović et al. 2005). Its main idea is to use several formulations of a considered problem and to switch from one to another formulation, depending on the current state of the solution process. When the change from one formulation to

Algorithm 20: The RECIPE heuristic for solving MINLP

```
Function RECIPE (k_max, t_max, x*)
1  t ← 0
2  while t < t_max do
3  |  k ← 1
4  |  while k ≤ k_max do
5  |  |  i ← 1
6  |  |  while i ≤ b do
7  |  |  |  Sample x̄ ∈ N_k(x*) at random
8  |  |  |  x ← SQP(x̄)
9  |  |  |  if x not feasible then
10 |  |  |  |  x ← x̄
   |  |  |  end
11 |  |  |  x' ← B&B (x, k, k_max)
12 |  |  |  if x' is better than x* then
13 |  |  |  |  x* ← x'; k ← 0; break;
   |  |  |  end
   |  |  |
14 |  |  |  i ← i + 1
   |  |  end
   |  |
15 |  |  k ← k + 1
   |  end
   |
16 |  t ← CpuTime()
   end
17 return x*;
```

another occurs, the solution resulting from the former formulation is used as an initial solution for the latter formulation. In Algorithm 21, a formulation change procedure is given. In this procedure, notation $f(\phi, x)$ corresponds to the objective value of the solution x in the formulation ϕ, while assignment $\phi \leftarrow \phi'$ represents the change of a formulation.

One variant of variable formulation space search is the so-called *variable formulation search*. It uses alternative formulations of the problem to determine which solution is more promising when they have the same value of the objective function in the original formulation. Variable formulation search was applied to the Cutwidth Minimization Problem in Pardo et al. (2013).

Algorithm 21: Formulation change procedure

```
Procedure Formulation_change(x, x', φ, φ',k)
if f(φ', x') < f(φ, x) then
|  x ← x'; φ ← φ'; k ← 1;
else
|  k ← k + 1;
end
```

4.6 Matheuristics with VNS

Variable neighborhood search has been combined with solvers for mixed integer programs (MIP) in the context of providing a first feasible solution as well as finding good quality solutions. Here, we present such heuristics used to tackle 0–1 mixed integer programs. A 0–1 mixed integer program (MIP) may be written in the following form:

$$(MIP)\begin{cases}\min\ v = cx \\ s.t.\ Ax \le b \\ 0 \le x_j \le U_j, \quad j \in V = \{1, \ldots, n\} \\ x_j \in \{0, 1\}, \quad j \in \mathcal{I} \subseteq V\end{cases}, \tag{3}$$

where A is a $m \times n$ constant matrix, b is a constant vector and the set V denotes the index set of variables, while the set $\mathcal{I}$ contains indices of binary variables. Each variable x_j has an upper bound denoted by U_j (which equals 1 if x_j is a binary variable, while otherwise it may be infinite). The integer problem defined in this manner will be denoted simply by MIP, while the relaxation of MIP obtained by excluding integrality constraints will be denoted by LP. A feasible solution of $MIP(LP)$ will be called $MIP(LP)$ feasible. An optimal solution of the LP problem will be denoted by $\overline{x}$. The set of all MIP feasible solutions will be denoted by X, i.e., $X = \{x \in \mathbb{R}^n : Ax \le b;\ 0 \le x_j \le U_j,\ j \in V = \{1, \ldots, n\};\ x_j \in \{0, 1\},\ j \in \mathcal{I} \subseteq V\}$. An optimal solution or the best found solution obtained in an attempt to solve the MIP problem will be denoted by x^*, while its objective value will be denoted by v^*.

Hamming distance between two solutions x and x' such that $x_j, x'_j \in \{0, 1\},\ j \in \mathcal{I}$ is defined by

$$\delta(x, x') = \sum_{j \in \mathcal{I}} |x_j - x'_j| = \sum_{j \in \mathcal{I}} x_j(1 - x'_j) + x'_j(1 - x_j).$$

The partial Hamming distance between x and x', relative to the subset $J \subset \mathcal{I}$, is defined as $\delta(J, x, x') = \sum_{j \in J} | x_j - x'_j |$ (obviously, $\delta(\mathcal{I}, x, x') = \delta(x, x')$).

The MIP relaxation of the 0-1 MIP problem relative to a subset $J \subset \mathcal{I}$ is expressed as:

$$(MIP(J))\begin{cases}\min\ v = cx \\ s.t.\ Ax \le b \\ 0 \le x_j \le U_j, \quad j \in V = \{1, \ldots, n\} \\ x_j \in \{0, 1\}, \quad j \in J \subset \mathcal{I} \subseteq V\end{cases}. \tag{4}$$

Given an MIP problem, $P\ \min\{cx \mid x \in X\}$ and an arbitrary LP feasible solution x^0. The problem *reduced* from the original problem P and associated with x^0 and a subset $J \subseteq \mathcal{I}$ is defined as:

$$P(x^0, J)\ \min\{cx \mid x \in X, x_j = x_j^0 \text{ for } j \in J \text{ such that } x_j^0 \in \{0, 1\}\}. \tag{5}$$

Note that in the case that $J = \mathcal{I}$, the reduced problem will be denoted by $P(x^0)$.

Similarly, given an MIP problem P and a solution $\widetilde{x}$ such that $\widetilde{x}_j \in \{0, 1\}, \ j \in \mathcal{I}$. Then, $MIP(P, \widetilde{x})$ will denote a minimization problem, obtained from the MIP problem P by replacing the original objective function with $\delta(x, \widetilde{x})$:

$$MIP(P, \tilde{x}) \qquad \min\{\delta(\tilde{x}, x) \mid x \in X\}. \tag{6}$$

The LP relaxation of such defined MIP problem $MIP(P, \tilde{x})$ will be denoted by $LP(P, \tilde{x})$.

If C is a set of constraints, we will denote with $(P \mid C)$ the problem obtained by adding all constraints in C to the problem P.

Let α be a real number, then $near(\alpha)$ will refer to the nearest integer value of a real value α, i.e., $near(\alpha) = \lfloor \alpha + 0.5 \rfloor$, where $\lfloor \alpha + 0.5 \rfloor$ represents the integer part of the number $\alpha + 0.5$ (i.e., the greatest integer $\leq \alpha + 0.5$). Furthermore, let x be a vector, then $near(x)$ will represent the nearest integer vector relative to the vector x, by defining each component as $near(x)_j = near(x_j) = \lfloor x_j + 0.5 \rfloor$.

4.6.1 Variable neighborhood branching

In 2006, Hansen et al. (2006) proposed variable neighborhood search (VNS) heuristic combined with local branching (LB) (Fischetti and Lodi 2003), called *variable neighborhood branching* (see Algorithm 22), for solving mixed-integer programs which may be seen as a generalization of local branching. The main advantage of the proposed VNS compared to the LB heuristic is the fact that it performs more systematic neighborhood exploration than local branching (Fischetti and Lodi 2003).

Algorithm 22: Variable neighborhood descent branching

Function `VND-MIP`$(P, t_{vnd}, t_{mip}, r_{max}, x')$;
Set $r = 1, t_{start} = CpuTime(), t = 0$;
Set $Q = P$;
while $t < t_{vnd}$ *and* $r \leq r_{max}$ **do**
  set $time\_limit = min\{t_{mip}, t_{vnd} - t\}$;
  $Q = (Q | \{\delta(x', x) \leq r\})$;
  $x'' =$ `MIPsolve`$(Q, time\_limit, x')$;
  switch `solution status` **do**
    case `OptSolFound`:
      Reverse last pseudo-cut into $\delta(x', x) > r + 1$;
      $x' = x'', r = 1$;
    case `feasibleSolFound`:
      Replace last pseudo-cut with $\delta(x', x) \geq 1$;
      $x' = x'', r = 1$;
    case `ProvenInfeasible`:
      Reverse last pseudo-cut into $\delta(x', x) > r + 1$;
      $r = r + 1$;
    case `nofeasiblesolfound`:
      return x';
  endsw
  set $t_{end} = CpuTime(), t = t_{end} - t_{start}$;
end
return x';

4.6.2 Hybrid variable neighborhood decomposition search heuristics

In 2010, Lazić et al. (2010) proposed a hybrid heuristic for solving 0–1 mixed integer programs which combines variable neighborhood decomposition search (VNDS) with the CPLEX MIP solver (see Algorithm 23). The algorithm starts solving the LP-relaxation of the original problem, obtaining an optimal solution $\bar{x}$. If the optimal solution $\bar{x}$ is integer feasible, the procedure returns $\bar{x}$ as an optimal solution of the initial problem. Otherwise, an initial feasible solution x is generated. At each iteration of the VNDS procedure, the distances $\delta_j = |x_j - \bar{x}_j|$ between the current incumbent solution values and corresponding LP-relaxation solution values are computed. Those distance values serve as criteria of choosing variables which will be fixed. Namely, at each iteration, k variables, whose indices correspond to the indices of k smallest δ_j values, are fixed at their values in the current incumbent solution x. After that, the resulting problem is solved using the CPLEX MIP solver. If an improvement of the current solution is achieved, a variable neighborhood descent branching is launched as the local search in the whole solution space and the process is repeated. If not, the number of fixed variables in the current subproblem is decreased. The pseudo-code is given in Algorithm 23. The input parameters for the VNDS algorithm are: the MIP problem P; the parameter d, which controls the change of neighborhood size during the search process; parameters $t_{\max}, t_{\text{sub}}, t_{\text{vnd}}, t_{\text{mip}}, r_{\max}$, which represent the maximum running time allowed for VNDS, time allowed for solving subproblems, time allowed for call to the `VND-MIP` procedure and time allowed for call to the MIP solver within the `VND-MIP` procedure, respectively. Finally, the parameter $r_{\max}$ represents the maximum size of the neighborhood to be explored within the `VND-MIP` procedure. In the pseudo-code, the statement of the form $y = \texttt{FindFirstFeasible}(P)$ denotes a call to a generic MIP solver, an attempt to find a first feasible solution of an input problem P. Further, the statement of the form $y = \texttt{MIPsolve}(P, t, x^*)$ denotes a call to a generic MIP solver to solve input problem P within a given time limit t starting from the best solution found x^*.

Algorithm 23: Variable neighborhood decomposition search for 0-1 MIP

Function VNDS($P, d, t_{max}, t_{sub}, t_{vnd}, t_{mip}, r_{max}$);
Solve the LP relaxation of P to obtain an optimal LP basic solution $\bar{x}$;
if $\bar{x}$ MIP feasible **then return** $\bar{x}$;
$x^* =$ FindFirstFeasible(P);
set $t_{start} = CpuTime(); t = 0;$
while $t < t_{max}$ **do**
  $\delta_j =\mid x_j^* - \bar{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}, j = 1, \ldots, |\mathcal{I}| - 1;$
  set $q =\mid \{j \in \mathcal{I} \mid \delta_j \neq 0\} \mid$;
  set $k_{step} = near(q/d), k = |\mathcal{I}| - k_{step};$
  while $t < t_{max}$ *and* $k > 0$ **do**
    $x' =$ MIPsolve($P(\tilde{x}, \{1, \ldots, k\}), t_{sub}, x^*$) ;
    if $cx' < cx^*$ **then**
      $x^* =$ VND-MIP($P, t_{vnd}, t_{mip}, r_{max}, x'$);
      break;
    else
      if $k - k_{step} > |\mathcal{I}| - q$ **then** $k_{step} = \max\{near(k/2), 1\};$
      set $k = k - k_{step}$;
      set $t_{end} = CpuTime(), t = t_{end} - t_{start};$
    end
  end
end
return x^*;

In 2010, Hanafi et al. (2010) proposed a hybrid variable neighborhood decomposition search heuristic that constitutes an improved version of variable neighborhood decomposition search heuristic proposed in Lazić et al. (2010). The enhancement is achieved by restricting the search space by adding pseudo-cuts, to avoid multiple explorations of the same areas. A sequence of lower and upper bounds on the problem objective is produced by adding pseudo-cuts, thereby reducing the integrality gap.

4.6.3 Variable neighborhood pump

In 2010, Hanafi et al. (2010) proposed a new method for finding an initial feasible solution for mixed integer programs called *Variable Neighborhood Pump* (VNP) (Algorithm 24), which combines variable neighborhood branching (VNB) (Hansen et al. 2006) and feasibility pump heuristics (Fischetti et al. 2005). The VNP works in the following way. First, an optimal solution $\bar{x}$ of the LP-relaxation of the initial 0–1 MIP problem is determined. After that, the obtained solution is rounded and one iteration of the FP pumping cycle is applied on it to obtain a near-feasible vector $\tilde{x}$. Then on the solution $\tilde{x}$, variable neighborhood branching, adapted for 0–1 MIP feasibility (Hanafi et al. 2010), is applied, in an attempt to locate a feasible solution of the original problem. If VNB does not return a feasible solution, a pseudo-cut is added to the current subproblem to change the linear relaxation solution, and the process is iterated. VNB returns either a feasible solution or reports failure and returns the last integer (infeasible) solution.

Algorithm 24: Variable neighborhood pump for 0-1 MIP

Function VNP(P);
Set $proceed1 = \texttt{true}$;
while $proceed1$ **do**
  Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
  Set $\tilde{x} = near(\overline{x})$;
  Set $proceed2 = \texttt{true}$;
  while $proceed2$ **do**
    if $\overline{x}$ *is integer* **then return** $\overline{x}$;
    Solve the $LP(P, \tilde{x})$ problem to obtain an optimal solution $\overline{x}$;
    if $\tilde{x} \neq near(\overline{x})$ **then**
      $\tilde{x} = near(\overline{x})$;
    else
      Set $proceed2 = \texttt{false}$;
    end
  end
  $k_{min} = \lfloor \delta(\tilde{x}, \overline{x}) \rfloor$; $k_{max} = \lfloor (|\mathcal{I}| - k_{min})/2 \rfloor$; $k_{step} = (k_{max} - k_{min})/5$;
  $x' = \text{VNB}(P, \tilde{x}, k_{min}, k_{step}, k_{max})$;
  if $x' = \tilde{x}$ **then**
    $P = (P \mid \delta(x, \overline{x}) \geq k_{min})$; Update $proceed1$;
  else
    return x';
  end
end
Output message: "No feasible solution found"; **return** $\tilde{x}$;

4.6.4 Diving heuristics

In 2014, Lazić et al. (2014) proposed two diving heuristics for obtaining a first MIP feasible solution. Diving heuristics are based on the systematic hard variable fixing (diving) process, according to the information obtained from the linear relaxation solution of the problem. They rely on the observation that a general-purpose MIP solver can be used not only for finding (near) optimal solutions of a given input problem, but also for finding the initial feasible solution.

The variable neighborhood (VN) diving algorithm begins by obtaining the LP-relaxation solution $\overline{x}$ of the original problem P and generating an initial integer (not necessarily feasible) solution $\tilde{x} = near(\overline{x})$ by rounding the LP-solution $\overline{x}$. If the optimal solution $\overline{x}$ is integer feasible for P, VN diving stops and returns $\overline{x}$. At each iteration of the VN diving procedure, the distances $\delta_j = |\tilde{x}_j - \overline{x}_j|$ from the current integer solution values $(\tilde{x}_j)_{j \in \mathcal{I}}$ to the corresponding LP-relaxation solution values $(\overline{x}_j)_{j \in \mathcal{I}}$ are computed and the variables $\tilde{x}_j$, $j \in \mathcal{I}$ are indexed so that $\delta_1 \leq \delta_2 \leq \cdots \leq \delta_{|\mathcal{I}|}$. Then, VN diving successively solves the subproblems $P(\tilde{x}, \{1, \ldots, k\})$ obtained from the original problem P, where the first k variables are fixed to their values in the current incumbent solution $\tilde{x}$. If a feasible solution is found by solving $P(\tilde{x}, \{1, \ldots, k\})$, it is returned as a feasible solution of the original problem P. Otherwise, a pseudo-cut $\delta(\{1, \ldots, k\}, \tilde{x}, x) \geq 1$ is added to avoid exploring the search space of $P(\tilde{x}, \{1, \ldots, k\})$ again and the next subproblem is examined. If no feasible solution is detected after solving all subproblems $P(\tilde{x}, \{1, \ldots, k\})$, $k_{\min} \leq k \leq k_{\max}$, $k_{\min} = k_{\text{step}}$, $k_{\max} = |\mathcal{I}| - k_{step}$, the linear relaxation of the current

problem P, which includes all the pseudo-cuts added during the search process, is solved and the process is iterated. If no feasible solution has been found due to the fulfillment of the stopping criteria, the algorithm reports failure and returns the last (infeasible) integer solution.

The pseudo-code of the VN diving heuristic is given in the Algorithm 25. The input parameters for the VN diving algorithm are the input MIP problem P and the parameter d, which controls the change of neighborhood size during the search process. In the pseudo-code, the statement of the form $y = \texttt{FindFirstFeasible}(P, t)$ denotes a call to a generic MIP solver, an attempt to find a first feasible solution of an input problem P within a given time limit t. If a feasible solution is found, it is assigned to the variable y, otherwise y retains its previous value.

Algorithm 25: Variable neighborhood diving for 0-1 MIP feasibility

```
   Function VNdiving(P, d);
1  Set proceed1 = true, proceed2 = true; Set timeLimit for subproblems;
2  while proceed1 do
3      Solve the LP relaxation of P to obtain an optimal LP basic solution x̄;
4      x̃ = near(x̄);
5      if x̄ = x̃ then return x̃;
6      δ_j =| x̃_j − x̄_j |; index x_j so that δ_j ≤ δ_{j+1}, j = 1, ..., |I| − 1;
7      Set n_d =| {j ∈ I | δ_j ≠ 0} |, k_step = near(n_d/d), k = |I| − k_step;
8      while proceed2 and k ≥ 0 do
9          J_k = {1, ..., k}; x' = FindFirstFeasible(P(x̃, J_k), timeLimit);
10         if P(x̃, J_k) is proven infeasible then P = (P | δ(J_k, x̃, x) ≥ 1);
11         if x' is feasible then return x';
12         if k − k_step > |I| − n_d then k_step = max{near(k/2), 1};
13         Set k = k − k_step;
14         Update proceed2;
       end
15     Update proceed1;
   end
16 Output message: "No feasible solution found"; return x̃;
```

In the case of variable neighborhood diving, a set of subproblems $P(\tilde{x}, J_k)$, for different values of k, is examined in each iteration until a feasible solution is found. In the single neighborhood diving procedure, we only examine one subproblem $P(\tilde{x}, J_k)$ in each iteration (a single neighborhood, see Algorithm 26). However, because only a single neighborhood is examined, additional diversification mechanisms are required. This diversification is provided through keeping the list of constraints which ensures that the same reference integer solution $\widetilde{x}$ cannot occur more than once (i.e., in more than one iteration) in the solution process. An additional MIP problem Q is introduced to store these constraints. In the beginning of the algorithm, Q is initialized as an empty problem (see line 4 in Algorithm 26). Then, in each iteration, if the current reference solution $\widetilde{x}$ is not feasible (see line 8 in Algorithm 26), constraint $\delta(\widetilde{x}, x) \geq \lceil \delta(\widetilde{x}, \overline{x}) \rceil$ is added to Q (line 9). This guarantees that future reference solutions cannot be the same as the current one, since the next reference solution is obtained by solving the problem MIP$(Q, near(\overline{x}))$ (see line 17), which contains all constraints from Q [see definition (6)]. The variables to be fixed in the current subproblem are chosen among those which have the same value as in the linear relaxation solution of the modified

problem LP($\widetilde{x}$), where $\widetilde{x}$ is the current reference integer solution (see lines 7 and 11). The number of variables to be fixed is controlled by the parameter α (line 11). After initialization (line 5), the value of α is updated in each iteration, depending on the solution status returned from the MIP solver. If the current subproblem is proven infeasible, the value of α is increased to reduce the number of fixed variables in the next iteration (see line 16) and thus provide better diversification. Otherwise, if the time limit allowed for the subproblem is exceeded without reaching a feasible solution or proving the subproblem infeasibility, the value of α is decreased. Decreasing the value of α, increases the number of fixed variables in the next iteration (see line 17) and thus reduces the size of the next subproblem. In the feasibility pump, the next reference integer solution is obtained by simply rounding the linear relaxation solution $\overline{x}$ of the modified problem LP($\widetilde{x}$). However, if $near(\overline{x})$ is equal to some of the previous reference solutions, the solution process is caught in a cycle. To avoid this type of cycling, we determine the next reference solution as the one which is at the minimum distance from $near(\overline{x})$ (with respect to binary variables) and satisfies all constraints from the current subproblem Q (see line 18). In this way, the convergence of the variable neighborhood diving algorithm is guaranteed (see Lazić et al. 2014).

Algorithm 26: Single neighborhood diving for 0-1 MIP feasibility

Function `SNDiving`(P);
1 Solve the LP relaxation of P to obtain an optimal LP basic solution $\overline{x}$;
2 Set $i = 0$; Set $\widetilde{x}^0 = near(\overline{x})$;
3 **if** ($\overline{x} = \widetilde{x}^0$) **then return** $\widetilde{x}^0$;
4 Set $Q_0 = \emptyset$;
5 Set *proceed* = `true`; Set *timeLimit* for subproblems; Set value of α;
6 **while** *proceed* **do**
7 Solve the $LP(P, \widetilde{x}^i)$ problem to obtain an optimal solution $\overline{x}$;
8 **if** ($\lceil \delta(\widetilde{x}^i, \overline{x}) \rceil = 0$) **then return** $\widetilde{x}^i$;
9 $Q_{i+1} = (Q_i \mid \delta(\widetilde{x}^i, x) \geq \lceil \delta(\widetilde{x}^i, \overline{x}) \rceil)$;
10 $\delta_j = \mid \tilde{x}^i_j - \overline{x}_j \mid$; index x_j so that $\delta_j \leq \delta_{j+1}$, $j = 1, \ldots, |\mathcal{I}| - 1$;
11 $k = near(\mid \{j \in \mathcal{I} : \widetilde{x}^i_j = \overline{x}_j\} \mid /\alpha)$; $J_k = \{1, \ldots, k\}$;
12 $x' =$ `FindFirstFeasible`($P(\widetilde{x}^i, J_k)$, *timeLimit*);
13 **if** *feasible solution found* **then return** x';
14 **if** $P(\widetilde{x}^i, J_k)$ is proven infeasible **then**
15 $Q_{i+1} = (Q_{i+1} \mid \delta(J_k, \widetilde{x}^i, x) \geq 1)$; $P = (P \mid \delta(J_k, \widetilde{x}^i, x) \geq 1)$;
16 $\alpha = 3\alpha/2$;
else
17 **if** *time limit for subproblem exceeded* **then** $\alpha = \max\{1, \alpha/2\}$;
end
18 $\widetilde{x}^{i+1} =$ `FindFirstFeasible`(MIP($Q_{i+1}, near(\overline{x})$), *timeLimit*);
19 **if** MIP($Q_{i+1}, near(\overline{x})$) *is proven infeasible* **then** Output message: "Problem P is proven infeasible"; **return**;
20 $i = i + 1$;
end

Algorithms 22, 23, 24, 25 and 26 may look at first glance that they do not necessarily have VNS nature and therefore do not have strong connection with algorithms from Sects. 2 and 3. However, it should be noted that those methods all systematically use

different neighborhood structures induced from the Hamming distance. In Algorithm 22, the parameter r in fact represents the neighborhood index of VND algorithm from before. The same neighborhoods are used in Algorithm 23, where the size of the sub-problems is larger and larger, and the number of fixed variables is smaller and smaller. This is in fact VNDS from before, adapted for solving a 0–1 MIP problem using general-purpose solvers such as CPLEX and Gurobi. A similar analogy holds for Algorithms 24, 25 and 26.

4.7 Parallel VNS

Parallel VNS heuristics are usually applied for solving large instances of optimization problems in a reasonable time. In the literature, various parallelization strategies within VNS heuristics have been performed for this purpose. Besides straightforward parallelization strategies [e.g., (1) parallelize local search; (2) augment the number of solutions drawn from the current neighborhood and make a local search in parallel from each of them; or (3) do the same as (2), but update the information about the best solution found], several more sophisticated approaches that lead to much better performance have been proposed (see e.g., Davidovic and Crainic 2013). These parallel VNS heuristics were used to solve many combinatorial optimization problems (e.g., p-median problem, car sequencing problem, vehicle routing-related problems, job shop scheduling problems, dynamic memory allocation problems) (García-López et al. 2002; Crainic et al. 2004; Sevkli and Aydin 2007; Polacek et al. 2008; Knausz 2008; Pirkwieser and Raidl 2009; Yazdani et al. 2010; Todosijević et al. 2016; Sánchez-Oro et al. 2015) as well as multi-objective optimization problems (see e.g., Eskandarpour et al. 2013).

4.8 Hybrids

The VNS has been also hybridized with other metaheuristics [e.g., Tabu Search, Path Relinking, Simulated Annealing, Greedy Randomized Adaptive Search Procedures (GRASP), Genetic Algorithm, Particle Swarm Optimization, etc]. For example, Li et al. (2014) combined the VNS with the chemical-reaction optimization (CRO) and the estimation of distribution (EDA) to solve the hybrid flow shop scheduling problem. In Oliveira et al. (2015), general variable neighborhood search was hybridized with GRASP to solve the targeted offers problem in direct marketing campaigns. Belhaiza et al. proposed a hybrid variable neighborhood-Tabu Search heuristic for the vehicle routing problem with multiple time windows (Belhaiza et al. 2014), while Xiao et al. used variable neighborhood simulated annealing algorithm to solve capacitated vehicle routing problems. For other VNS hybridization, we refer the reader to Hansen et al. (2010).

5 Conclusions

The basic scheme of variable neighborhood search (VNS) along with its main ingredients (local search and shaking procedures) have been presented. More precisely, we

present most common local search procedures used within a VNS heuristic as well as a typical shaking procedure to resolve local optima traps. In addition, the paper contains VNS variants that have been deduced from the basic VNS scheme.

VNS-based heuristics turn out to be the state-of-the-art heuristics for many NP-hard optimization problems. Such performance indicates that developing VNS heuristics for solving other NP-hard optimization problems will lead to a promising research avenue. Thus, we believe that this paper will serve as a guide for developing new state-of-the-art heuristics based on VNS.

References

Audet C, Béchard V, Le Digabel S (2008) Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. J Global Optim 41(2):299–318

Audet C, Brimberg J, Hansen P, Digabel SL, Mladenović N (2004) Pooling problem: alternate formulations and solution methods. Manag Sci 50(6):761–776

Balas E, Martin C (1980) Pivot and complement-a heuristic for 0–1 programming. Manag Sci 26:86–96

Belhaiza S, Hansen P, Laporte G (2014) A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. Comput Oper Res 52:269–281

Bierlaire M, Thémans M, Zufferey N (2010) A heuristic for nonlinear global optimization. INFORMS J Comput 22(1):59–70

Brimberg J, Hansen P, Mladenovic N, Taillard ED (2000) Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem. Oper Res 48(3):444–460

Brimberg J, Mladenović N, Urošević D (2015) Solving the maximally diverse grouping problem by skewed general variable neighborhood search. Inf Sci 295:650–675

Bussieck MR, Drud AS, Meeraus A (2003) Minlplib—a collection of test models for mixed-integer nonlinear programming. INFORMS J Comput 15(1):114–119

Carrizosa E, Dražić M, Dražić Z, Mladenović N (2012) Gaussian variable neighborhood search for continuous optimization. Comput Oper Res 39(9):2206–2213

Carrizosa E, Mladenović N, Todosijević R (2013) Variable neighborhood search for minimum sum-of-squares clustering on networks. Eur J Oper Res 230(2):356–363

Crainic TG, Gendreau M, Hansen P, Mladenović N (2004) Cooperative parallel variable neighborhood search for the p-median. J Heuristics 10(3):293–314

Davidovic T, Crainic TG (2013) Parallelization strategies for variable neighborhood search. Tecnical report CIRRELT—2013–47

Di Gaspero L, Schaerf A (2006) Neighborhood portfolio approach for local search applied to timetabling problems. J Math Model Algorithms 5(1):65–89

de Armas J, Melián-Batista B (2015) Variable neighborhood search for a dynamic rich vehicle routing problem with time windows. Comput Indus Eng 85:120–131

Dražić M, Lavor C, Maculan N, Mladenović N (2008) A continuous variable neighbourhood search heuristic for finding the tridimensional structure of a molecule. Eur J Oper Res 185:1265–1273

Dražić M, Dražić Z, Mladenović N, Urošević D, Zhao QH (2014) Continuous variable neighbourhood search with modified nelder–mead for non-differentiable optimization. IMA J Manag Math. doi:10.1093/imaman/dpu012

Drazić M, Kovacevic-Vujcić V, Cangalović M, Mladenović N (2006) Glob—a new vns-based software for global optimization. In: Global optimization, Springer, pp 135–154

Eskandarpour M, Zegordi SH, Nikbakhsh E (2013) A parallel variable neighborhood search for the multi-objective sustainable post-sales network design problem. Int J Prod Econ 145(1):117–131

Fischetti M, Glover F, Lodi A (2005) The feasibility pump. Math Program 104(1):91–104

Fischetti M, Lodi A (2003) Local branching. Math Prog 98:23–47

Fletcher R, Leyffer S (1998) Numerical experience with lower bounds for miqp branch-and-bound. SIAM J Optim 8(2):604–616

García-López F, Melián-Batista B, Moreno-Pérez JA, Moreno-Vega JM (2002) The parallel variable neighborhood search for the p-median problem. J Heuristics 8(3):375–388

Gill PE, Murray W, Saunders MA (2002) Snopt: an sqp algorithm for large-scale constrained optimization. SIAM J Optim 12(4):979–1006

Glover F, McMillan C, Glover R (1984) A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. J Oper Manag 4(2):113–128

Hanafi S, Lazić J, Mladenović N (2010) Variable neighbourhood pump heuristic for 0–1 mixed integer programming feasibility. Electron Notes Discrete Math 36:759–766

Hanafi S, Lazić J, Mladenović N, Wilbaut C, Crévits I (2010) Hybrid variable neighbourhood decomposition search for 0–1 mixed integer programming problem. Electron Notes Discrete Math 36:883–890

Hansen P, Brimberg J, Uroševic D, Mladenovic N (2007) Primal-dual variable neighborhood search for the simple plant-location problem. INFORMS J Comput 19(4):552–564

Hansen P, Mladenović N (2006) First vs. best improvement: an empirical study. Discrete Appl Math 154(5):802–817

Hansen P, Mladenović N, Pérez JAM (2010) Variable neighbourhood search: methods and applications. Ann Oper Res 175(1):367–407

Hansen P, Mladenović N, Perez-Britos D (2001) Variable neighborhood decomposition search. J Heuristics 7(4):335–350

Hansen P, Mladenović N, Urosević D (2006) Variable neighborhood search and local branching. Comput Oper Res 33(10):3034–3045

Ilić A, Urošević D, Brimberg J, Mladenović N (2010) A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. Eur J Oper Res 206(2):289–300

Johnson DS, McGeoch L (1997) The traveling salesman problem: a case study in local optimization. In: Aarts E, Lenstra J (eds) Local search in combinatorial optimization. Wiley, New York, pp 215–310

Khemakhem M, Haddar B, Chebil K, Hanafi S (2012) A filter-and-fan metaheuristic for the 0–1 multidimensional knapsack problem. Int. J. Appl Metaheuristic Comput 3:43–63

Kirkpatrick S, Toulouse G (1985) Configuration space analysis of traveling salesman problems. J Phys 46:1277–1292

Kirlik G, Oguz C (2012) A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. Comput Oper Res 39(7):1506–1520

Knausz M (2008) Parallel variable neighbourhood search for the car sequencing problem. Tecnical report, Fakultat fur Informatik der Technischen Universitat Wien

Lazić J, Hanafi S, Mladenović N, Urosević D (2010) Variable neighbourhood decomposition search for 0–1 mixed integer programs. Comput Oper Res 37:1055–1067

Lazić J, Todosijević R, Hanafi S, Mladenović N (2014) Variable and single neighbourhood diving for mip feasibility. Yugoslav J Oper Res. doi:10.2298/YJOR140417027L

Li J, Pan Q, Wang F (2014) A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. Appl Soft Comput 24:63–77

Liberti L, Lavor C, Maculan N, Marinelli F (2009) Double variable neighbourhood search with smoothing for the molecular distance geometry problem. J Global Optim 43:207–218

Liberti L, Mladenović N, Nannicini G (2011) A recipe for finding good solutions to MINLPs. Math Program Comput 3:349–390

Liberti L, Nannicini G, Mladenović N (2010) A good recipe for solving minlps. In: Maniezzo V, Stuetze T, Voss S (eds) MATHEURISTICS: hybridizing metaheuristics and mathematical programming, Operations Research/Computer Science Interface Series. Springer, Berlin, pp 231–244

Lü Z, Hao JK, Glover F (2011) Neighborhood analysis: a case study on curriculum-based course timetabling. J Heuristics 17(2):97–118

Mjirda A, Todosijević R, Hanafi S, Hansen P, Mladenović N (2016) Sequential variable neighborhood descent variants: an empirical study on the traveling salesman problem. Int Trans Oper Res. doi:10.1111/itor.12282

Mladenović N, Dražić M, Kovačevic-Vujčić V, Čangalović M (2008) General variable neighborhood search for the continuous optimization. Eur J Oper Res 191(3):753–770

Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24(11):1097–1100

Mladenović N, Petrović J, Kovačević-Vujčić V, Čangalović M (2003) Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. Eur J Oper Res 151(2):389–399

Mladenović N, Plastria F, Urošević D (2005) Reformulation descent applied to circle packing problems. Comput Oper Res 32(9):2419–2434

Mladenović N, Todosijević R, Urošević D (2013) An efficient general variable neighborhood search for large travelling salesman problem with time windows. Yugoslav J Oper Res 23(1):19–31
Mladenović N, Todosijević R, Urošević D (2014) Two level general variable neighborhood search for attractive traveling salesman problem. Comput Oper Res 52:341–348
Mladenović N, Urošević D, Perez-Brito D (2014) Variable neighborhood search for minimum linear arrangement problem. Yugoslav J Oper Res. doi:10.2298/YJOR140928038M
Oliveira T, Coelho V, Souza MJF, Boava DLT, Boava F, Coelho IM, Coelho BN (2015) A hybrid variable neighborhood search algorithm for targeted offers in direct marketing. Electron Notes Discrete Math 47:205–212
Pardo EG, Mladenović N, Pantrigo JJ, Duarte A (2013) Variable formulation search for the cutwidth minimization problem. Appl Soft Comput 13(5):2242–2252
Pirkwieser S, Raidl GR (2009) Multiple variable neighborhood search enriched with ilp techniques for the periodic vehicle routing problem with time windows. In: Hybrid metaheuristics. Springer, pp 45–59
Polacek M, Benkner S, Doerner KF, Hartl RF (2008) A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows. BuR Bus Res 1(2):207–218
Rego C, Glover F (2010) Ejection chain and filter-and-fan methods in combinatorial optimization. Ann Oper Res 175(1):77–105
Reinelt G (1991) TSPLIB—a traveling salesman problem library. ORSA J Comput 3:376–384
Sánchez-Oro J, Sevaux M, Rossi A, Martí R, Duarte A (2015) Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies. Electron Notes Discrete Math 47:85–92
Sevkli M, Aydin ME (2007) Parallel variable neighbourhood search algorithms for job shop scheduling problems. IMA J Manag Math 18(2):117–133
Todosijević R, Benmansour R, Hanafi S, Mladenović N, Artiba A (2016) Nested general variable neighborhood search for the periodic maintenance problem. Eur J Oper Res 252(2):385–396
Todosijević R, Hanafi S, Urošević D, Jarboui B, Gendron B (2016) A general variable neighborhood search for the swap-body vehicle routing problem. Comput Oper Res. doi:10.1016/j.cor.2016.01.016
Todosijević R, Mjirda A, Mladenović M, Hanafi S, Gendron B (2014) A general variable neighborhood search variants for the travelling salesman problem with draft limits. Optim Lett. doi:10.1007/s11590-014-0788-9
Todosijević R, Mladenović M, Hanafi S, Crévits I (2012) Vns based heuristic for solving the unit commitment problem. Electron Notes Discrete Math 39:153–160
Todosijević R, Mladenović M, Hanafi S, Mladenović N, Crévits I (2016) Adaptive general variable neighborhood search heuristics for solving the unit commitment problem. Int J Electr Power Energy Syst 78:873–883
Todosijević R, Urošević D, Mladenović N, Hanafi S (2015) A general variable neighborhood search for solving the uncapacitated r-allocation p-hub median problem. Optim Lett. doi:10.1007/s11590-015-0867-6
Toksarı MD, Güner E (2007) Solving the unconstrained optimization problem by a variable neighborhood search. J Math Anal Appl 328(2):1178–1187
Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196(1):611–634
Yazdani M, Amiri M, Zandieh M (2010) Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Syst Appl 37(1):678–687
Zhao Q, Mladenovic N, Uroševic D (2012) A parametric simplex search for unconstrained optimization problem. Trans Adv Res 8:22–27
Zhao QH, Urosević D, Mladenović N, Hansen P (2009) A restarted and modified simplex search for unconstrained optimization. Comput Oper Res 36(12):3263–3271